Problem 7 Walkthrough

NFAs, Stage 2

Create an NFA for the language “all base 3 numbers without leading zeros that are divisible by 3”

Create an NFA for the language “all base 3 numbers without leading zeros that are divisible by 3”

Observe that in our typically base 10 number system, a number is divisible by 10 only if it ends in zero as well. Similarly, a base 3 number is divisible by 3 only if it ends in zero.

What our NFA **should** accept:
120
0
20

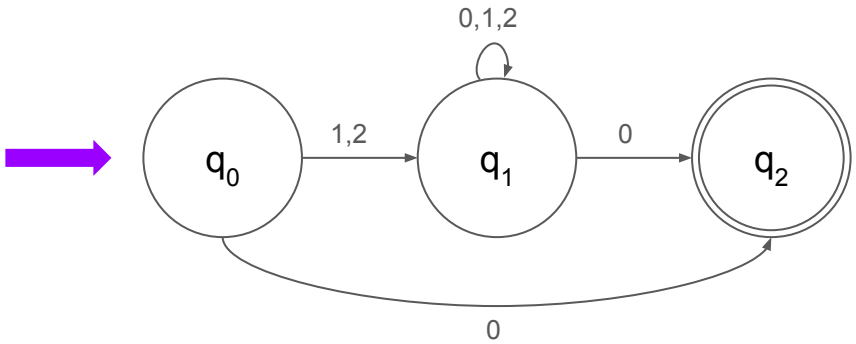
What our NFA **should not** accept:
1
02
101
00

Create an NFA for the language “all base 3 numbers without leading zeros that are divisible by 3”

Observe that in our typically base 10 number system, a number is divisible by 10 only if it ends in zero as well. Similarly, a base 3 number is divisible by 3 only if it ends in zero.

What our NFA **should** accept:
120
0
20

What our NFA **should not** accept:
1
02
101
00



Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

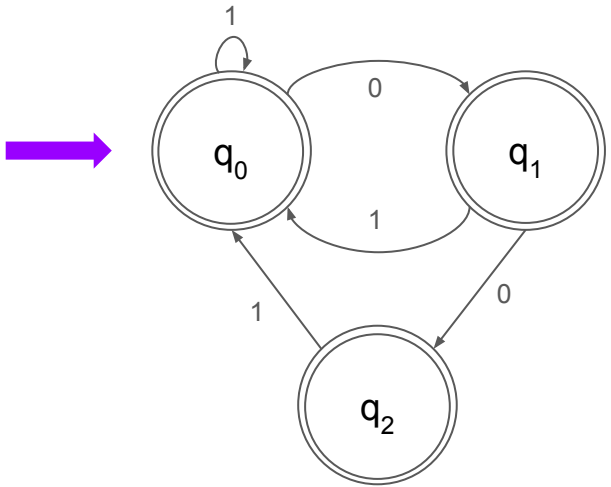
The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

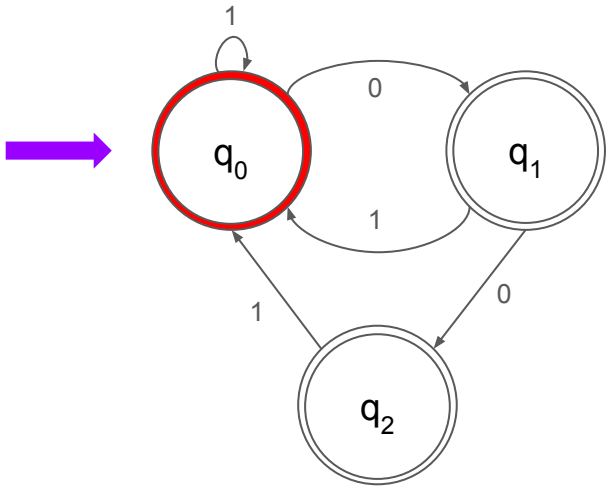


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 000

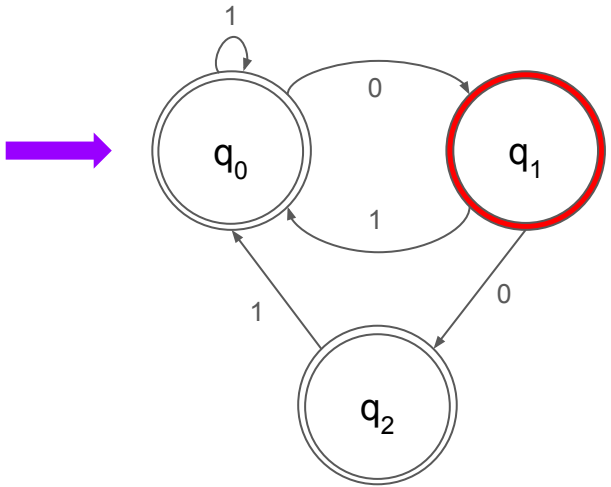


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 000

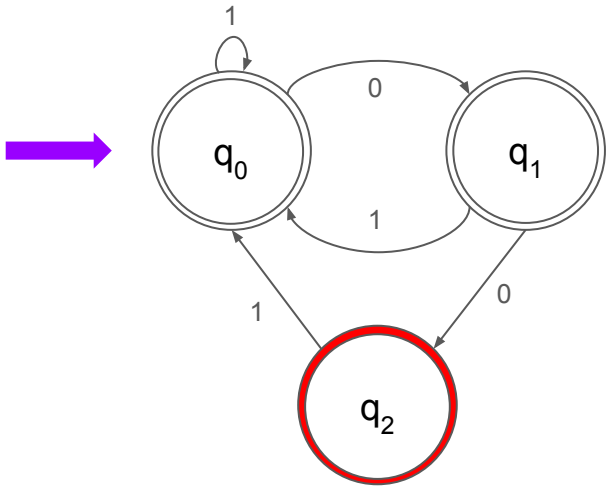


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 000

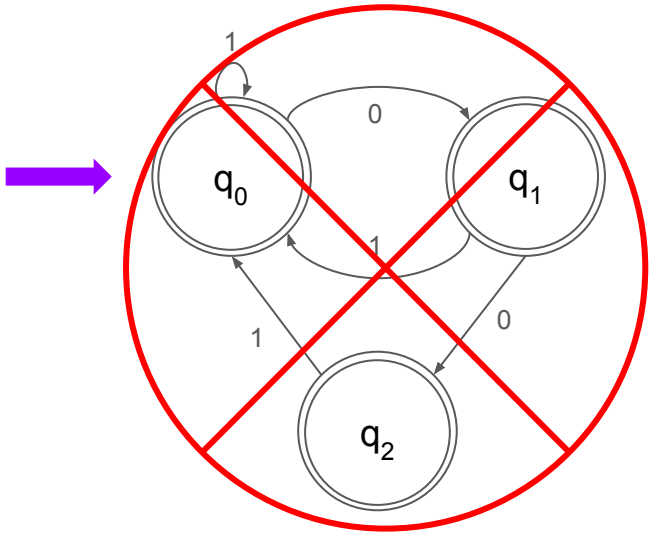


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 000

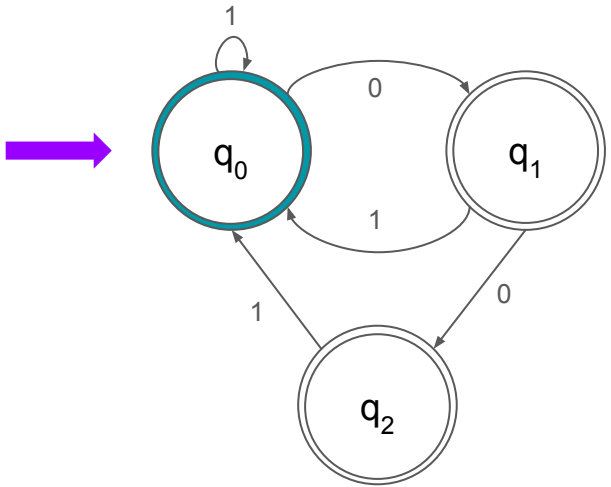


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 01001

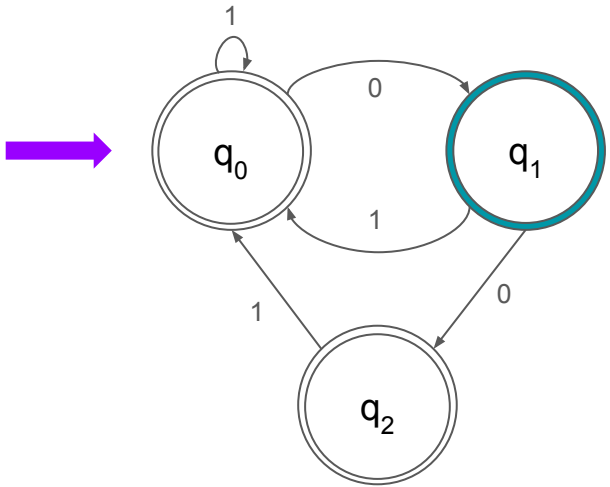


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 01001

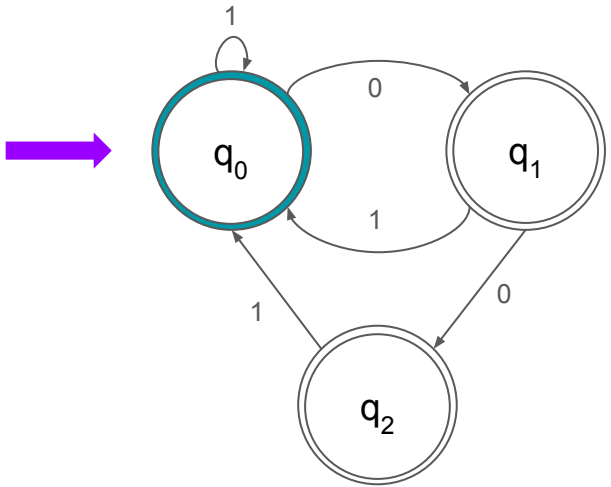


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 01001

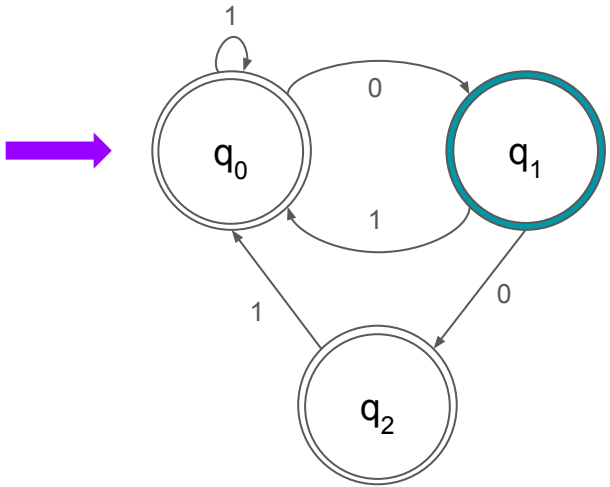


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 01001

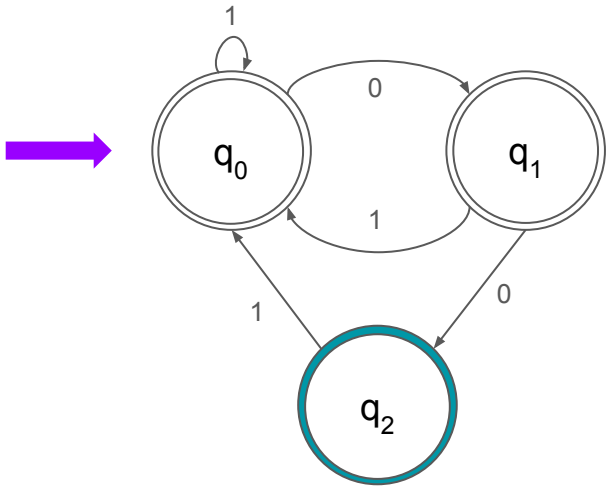


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 01001

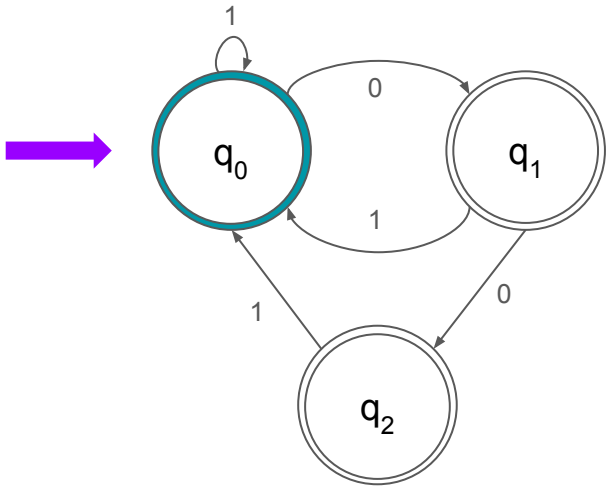


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

The strings in this language are of the form x111y where x and y are possibly empty binary strings that have at most two consecutive zeros.

Let’s first create NFAs for substrings x and y.

Example: 01001

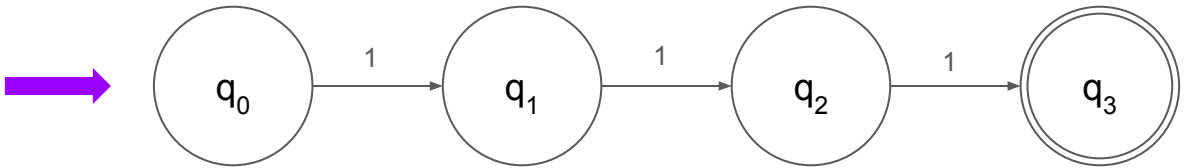


Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

How would we express the “111” in x**111**y?

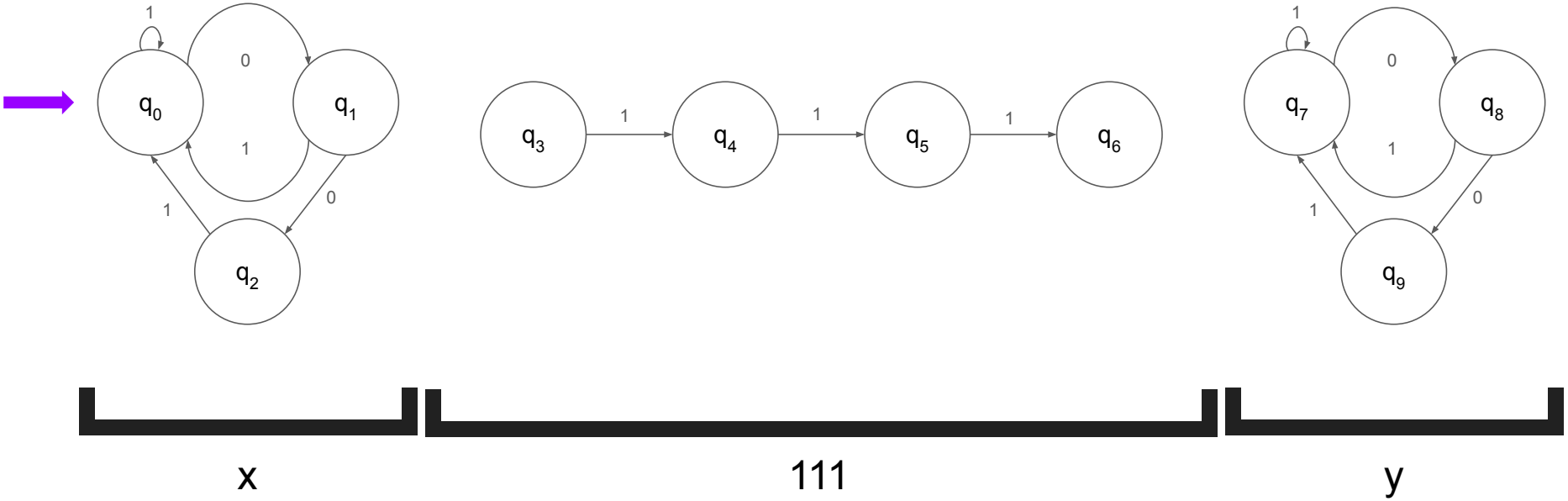
Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

How would we express the “111” in x**111**y?



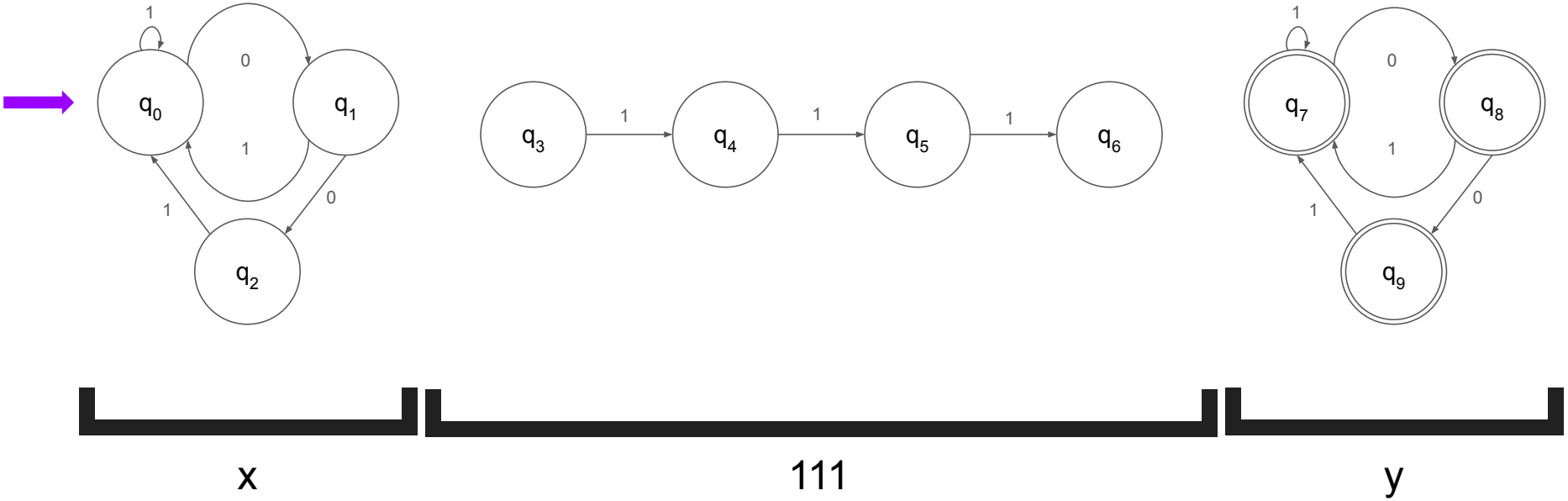
Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

Let’s start combining the NFAs!



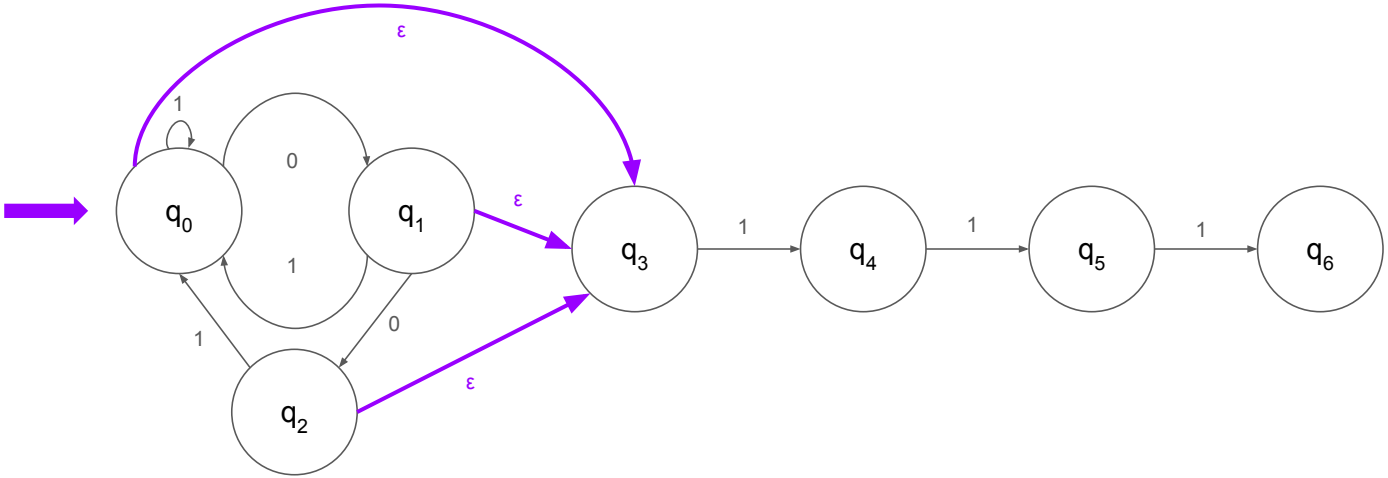
Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

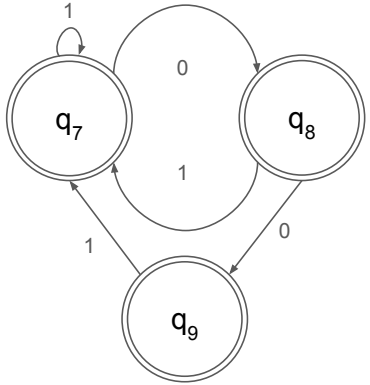
Let’s start combining the NFAs!



Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

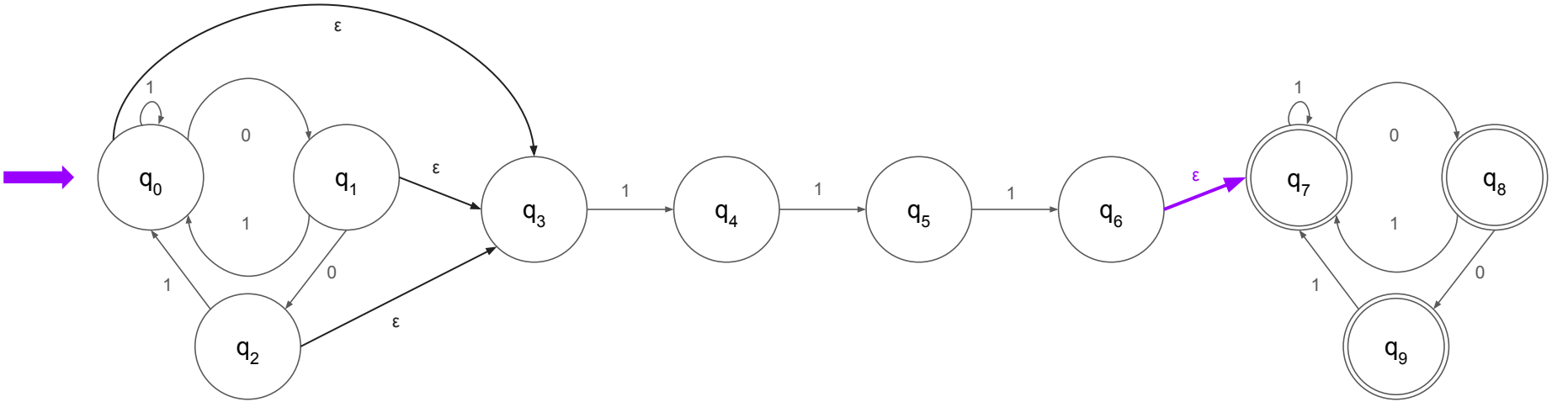
Let’s start combining the NFAs!





Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

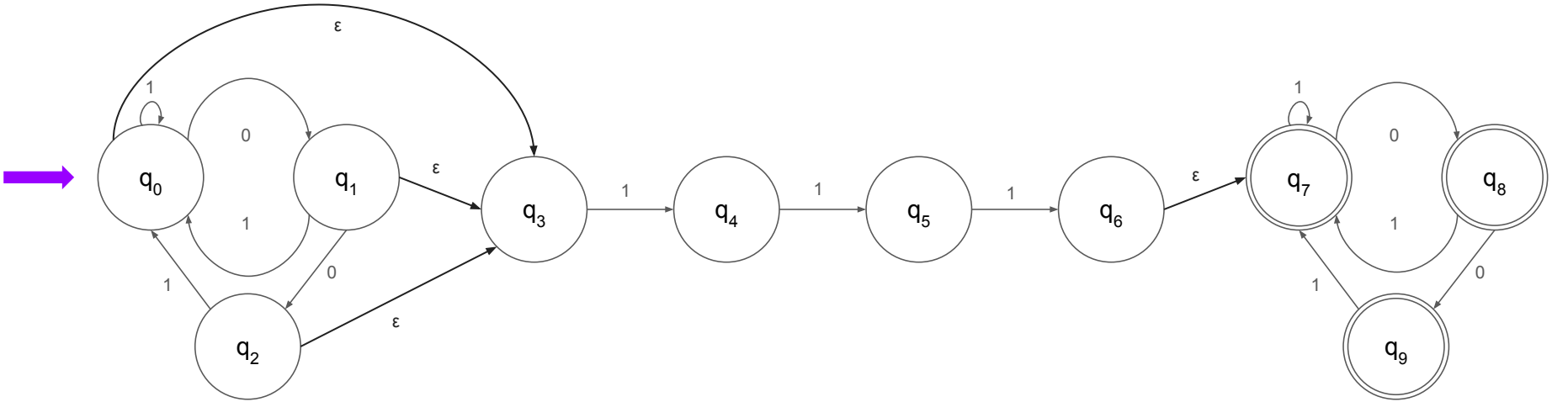
Let’s start combining the NFAs!



x 111 y

Create an NFA for the language “all binary strings that contain the substring ‘111’, but not the substring ‘000’”.

Let’s start combining the NFAs!



Create an NFA for the language “all binary strings that do not have any consecutive 0’s or 1’s”.

Create an NFA for the language “all binary strings that do not have any consecutive 0’s or 1’s”.

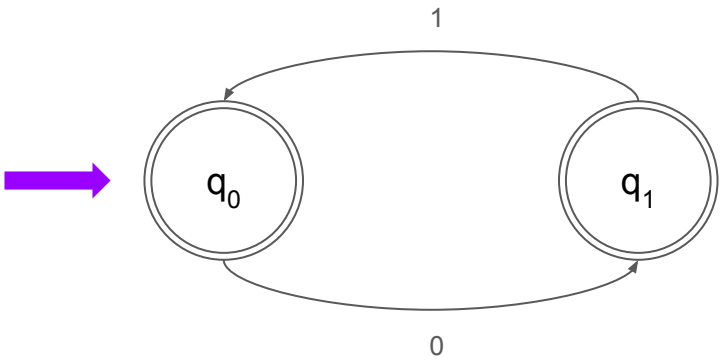
States: one representing the last seen character being a zero and another state representing the last seen character being a one.

Transitions: The only way to transition out of a state is by consuming the other character. As long as the entire string can be consumed, then both states should be accepting.

Create an NFA for the language “all binary strings that do not have any consecutive 0’s or 1’s”.

States: one representing the last seen character being a zero and another state representing the last seen character being a one.

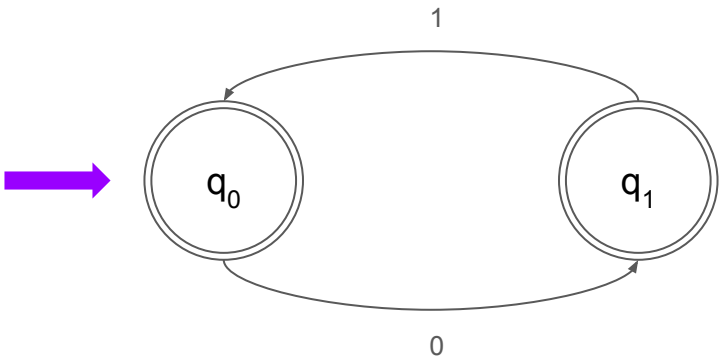
Transitions: The only way to transition out of a state is by consuming the other character. As long as the entire string can be consumed, then both states should be accepting.



Create an NFA for the language “all binary strings that do not have any consecutive 0’s or 1’s”.

States: one representing the last seen character being a zero and another state representing the last seen character being a one.

Transitions: The only way to transition out of a state is by consuming the other character. As long as the entire string can be consumed, then both states should be accepting.

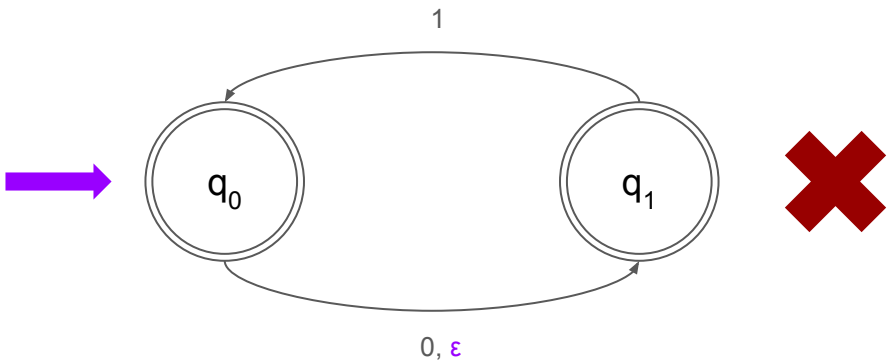


What if the string starts with a “1”?

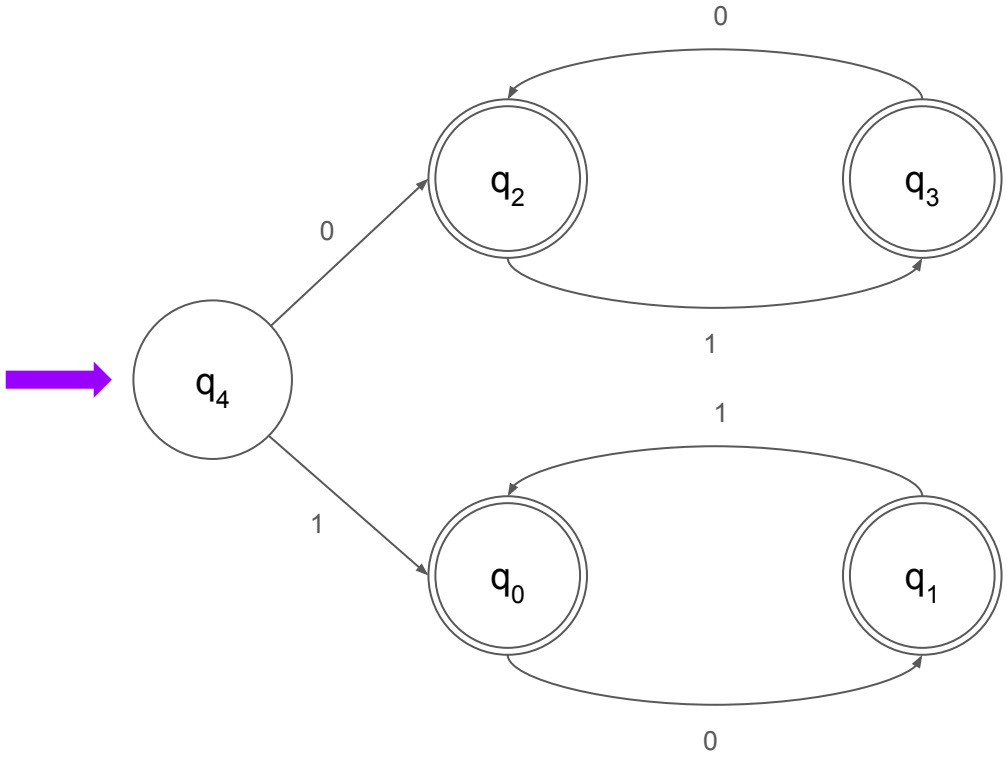
Create an NFA for the language “all binary strings that do not have any consecutive 0’s or 1’s”.

States: one representing the last seen character being a zero and another state representing the last seen character being a one.

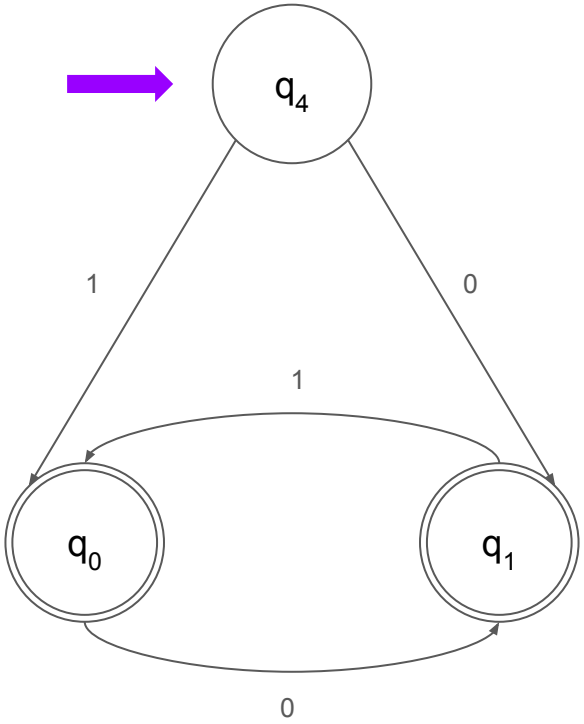
Transitions: The only way to transition out of a state is by consuming the other character. As long as the entire string can be consumed, then both states should be accepting.



Create an NFA for the language “all binary strings that do not have any consecutive 0’s or 1’s”.



Create an NFA for the language “all binary strings that do not have any consecutive 0’s or 1’s”.



Strategies / Intuition for building NFAs

- Observe some property about the language and “force” that property in our NFA.
  - In our first problem with base 3, we observed that all accepted strings must end in zero. We force that property in our NFA by ensuring that zero is the last character consumed.
- Build NFAs for substrings and combine.
  - In our second problem, we decomposed our language into x111y and built NFAs for x, y, and 111 before recombining them for our original language.
  - **Be careful when combining!**
- Enumerate states and transitions
  - In our third problem, we enumerated what possible states and transitions that we might encounter when consuming our strings in our language.

Strategies / Intuition for building NFAs

- Observe some property about the language and “force” that property in our NFA.
  - In our first problem with base 3, we observed that all accepted strings must end in zero. We force that property in our NFA by ensuring that zero is the last character consumed.
- Build NFAs for substrings and combine.
  - In our second problem, we decomposed our language into x111y and built NFAs for x, y, and 111 before recombining them for our original language.
  - **Be careful when combining!**
- Enumerate states and transitions
  - In our third problem, we enumerated what possible states and transitions that we might encounter when consuming our strings in our language.

Strategies / Intuition for building NFAs

- Observe some property about the language and “force” that property in our NFA.
  - In our first problem with base 3, we observed that all accepted strings must end in zero. We force that property in our NFA by ensuring that zero is the last character consumed.
- Build NFAs for substrings and combine.
  - In our second problem, we decomposed our language into x111y and built NFAs for x, y, and 111 before recombining them for our original language.
  - **Be careful when combining!**
- Enumerate states and transitions
  - In our third problem, we enumerated what possible states and transitions that we might encounter when consuming our strings in our language.